## REMARKS

This paper is being provided in response to the Final Office Action dated July 19, 2007, for the above-referenced application. In this response, Applicants have amended claims to clarify that which Applicants consider to be the claimed invention. Applicant respectfully submits that the amendments to the claims are fully supported by the originally filed application.

The objection to claim 19 for informalities has been addressed by amendments contained here in accordance with the guidelines set forth in the Office Action. Accordingly, Applicants respectfully request that the objection be reconsidered and withdrawn.

The rejection of claims 1, 3-4, 6-10, 16-17, 19-20, 22, 24-25, 27-31, 37-38, 40-41 under 35 U.S.C. 102(b) as being anticipated by U.S. Patent No. 5,574,898 to Leblang, et al. (hereinafter "Leblang") is hereby traversed and reconsideration thereof is respectfully requested in view of the amendments to the claims contained herein.

Independent claim 1, as amended herein, recites a computer implemented method for automatically tracking build information comprising: extracting build information by processing, for each of one or more builds, one or more software modules produced using a compilation process resulting in said one or more software modules, wherein said build information extracted includes software module information about at least one software module produced as an output of a compilation process for each of the one or more builds; registering the one or more builds by storing the build information corresponding to each of the one or more builds in a database;

automatically determining software module information about software being tested, wherein the software module information is gathered from one or more software modules during execution of the software being tested; performing a query of the database to retrieve first build information for a first build of the one or more builds in the database; and determining a volatility metric of code change that has occurred between software modules in both said first build information and said software module information of the software being tested, wherein said determining the volatility metric includes matching portions of the first build information to corresponding portions of said software module information and determining at least one of: a number of functions added, a number of functions removed and a number of functions modified in a software module of the software module information in comparison to a software module of the first build information. Claims 3-4, 6-15 depend directly or indirectly from independent claim 1.

Independent claim 16, as amended herein, recites a computer implemented method for determining a code volatility metric, the method comprising: extracting build information by processing, for at least two builds, one or more software modules produced using a compilation process resulting in the one or more software modules, wherein the build information extracted includes software module information about at least one software module produced as an output of a compilation process for each of the at least two builds; registering the at least two builds by storing the build information corresponding to each of said at least two builds in a database; identifying, by retrieving at least a portion of the build information from the database, a first and a second of the at least two builds; performing a query of the database to determine code volatility between software modules included in both the first and the second of the at least two builds; and calculating, in response to said query, the code volatility metric using the build

information including software module information about the first and the second builds included in the database, the code volatility metric being determined using one or more metrics representing an amount of code change that has occurred between software modules in both the first build and the second build, the one or more metrics including at least one of: a number of functions added, a number of functions removed and a number of functions modified in at least one software module of the second build in comparison to at least one software module of the first build. Claims 17 and 19 depend from independent claim 16.

Independent claim 20, as amended herein, recites a computer implemented method for tracking build information comprising: extracting build information by processing, for each of one or more builds, one or more software modules produced using a compilation process resulting in the one or more software modules, wherein the build information extracted includes software module information about at least one software module produced as an output of a compilation process for each of the one or more builds; registering the one or more builds by storing the build information in a database; executing a software program that includes one or more software modules, the executing including processing the one or more software modules of the software program during execution of the software program to determine module information about the one or more software modules of said software program; automatically determining, using the build information included in the database and the module information obtained from the executing, a matching build for the one or more software modules of the software program that are also associated with one of the builds previously registered; and determining testing information associated with the matching build by performing a query of the database, the testing information including at least one type of runtime analysis performed for the matching build, the

at least one type of runtime analysis including determining a volatility metric of code change that has occurred between software modules in both said matching build and said module information obtained from said executing, wherein said determining the volatility metric includes matching portions of the matching build to corresponding portions of the module information obtained from said executing and determining at least one of: a number of functions added, a number of functions removed and a number of functions modified in a software module of the module information obtained from said executing in comparison to a software module of the matching build.

Independent claim 22, as amended herein, recites a computer readable medium comprising machine executable code stored thereon for automatically tracking build information, the computer readable medium comprising: machine executable code for extracting build information by processing, for each of one or more builds, one or more software modules produced using a compilation process resulting in the one or more software modules, wherein said build information extracted includes software module information about at least one software module produced as an output of a compilation process for each of the one or more builds; machine executable code for registering the one or more builds by storing the build information corresponding to each of the one or more builds in a database; machine executable code for automatically determining software module information about software being tested, wherein the software module information is gathered from one or more software modules during execution of the software being tested; and machine executable code for determining a first of the one or more builds included in the database which corresponds to the software module information about the software being tested, the determining including determining a volatility

metric of code change that has occurred between software modules in both said one or more builds included in the database and said software module information about the software being tested, wherein said determining the volatility metric includes matching portions of the one or more builds included in the database to corresponding portions of the software module information about the software being tested and determining at least one of: a number of functions added, a number of functions removed and a number of functions modified in a software module of the software module information about the software being tested in comparison to a software module of the one or more builds included in the database. Claims 24, 25, 27-36 depend directly or indirectly from independent claim 22.

Independent claim 37, as amended herein, recites a computer readable medium comprising machine executable code stored thereon for determining a code volatility metric, the computer readable medium comprising: machine executable code for extracting build information by processing, for at least two builds, one or more software modules produced using a compilation process resulting in the one or more software modules, wherein the build information extracted includes software module information about at least one software module produced as an output of a compilation process for each of the at least two builds; machine executable code for registering the at least two builds by storing the build information corresponding to each of the at least two builds in a database; machine executable code for identifying, by retrieving at least a portion of the build information from the database, a first and a second of the at least two builds; machine executable code for performing a query of the database to determine code volatility between software modules included in both the first and the second of the at least two builds; and machine executable code for calculating, in response to

said query, said code volatility metric using the build information including software module information about the first and said second builds included in the database, the code volatility metric being determined using one or more metrics representing an amount of code change that has occurred between software modules in both the first build and said second build, the one or more metrics including at least one of: a number of functions added, a number of functions removed and a number of functions modified in at least one software module of the second build in comparison to at least one software module of the first build. Claims 38 and 40 depend directly or indirectly from independent claim 37.

Independent claim 41, as amended herein, recites a computer readable medium comprising executable code stored thereon for tracking build information, the computer readable medium comprising: machine executable code for extracting build information by processing, for each of one or more builds, one or more software modules produced using a compilation process resulting in said one or more software modules, wherein said build information extracted includes software module information about at least one software module produced as an output of a compilation process for each of the one or more builds; machine executable code for registering the one or more builds by storing said build information in a database; machine executable code for executing a software program that includes one or more software modules, the executing including processing the one or more software modules of the software program during execution of the software program to determine module information about the one or more software modules of the software program; machine executable code for automatically determining, using build information included in the database and the module information obtained from the executing, a matching build for the one or more software modules of the

software program that are also associated with one of said builds previously registered; and machine executable code for determining testing information associated with the matching build by performing a query of the database, the testing information including at least one type of runtime analysis performed for the matching build, the at least one type of runtime analysis including determining a volatility metric of code change that has occurred between software modules in both said matching build and said module information obtained from said executing, wherein said determining the volatility metric includes matching portions of the matching build to corresponding portions of the module information obtained from said executing and determining at least one of: a number of functions added, a number of functions removed and a number of functions modified in a software module of the module information obtained from said executing in comparison to a software module of the matching build.

Leblang's Figure 1 includes a version control system and stores of source objects and derived objects. Derived objects are created by running a system build process on a particular version of the source objects (See Figure 1, Col. 5, Lines 36-45). In Figure 7, the version control system is illustrated for use with two developers each using a different view having a set of versions from the versioned object bases (VOBs). (See Figures 2, 7; Col. 6, Lines 8-13; Col. 8, Line 51-Col. 10, Line 9). Source files are the input to the software build process. Equally important are the files that are created by software builds. With the version control system, such files are called derived objects. Each derived object has two main parts, the data itself and an associated configuration record. The configuration record is a "bill of materials" that stores an audit of the build that produced the derived object. This automatically includes a list of the element versions used in the build. It also includes versions of dependencies (such as build tools)

that are explicitly declared in the makefile. Derived objects, like elements, can be accessed either with standard UNIX pathnames or with version extended names. (Col. 25, Lines 27-41).

Leblang's version control system is used to maintain particular versions of source objects 24 used in producing builds. As described above, Leblang discloses a configuration record that includes a list of element versions used in the build and dependencies explicitly declared in the makefile. Leblang appears to use makefiles to determine the configuration record. Makefiles are used to specify build scripts with different inputs (e.g., source files) and outputs (e.g., binaries) generated. Leblang's system is used to maintain and track different versions of sources (see, for example, Figures 5 and 7). Applicants submit that Leblang discloses execution of build scripts, specifically execution of multiple makefile build scripts (see, for example, col. 2, lines 36-38 of Leblang), but appears silent regarding gathering software module information about software being tested during execution of software being tested. Further, Leblang appears to make no disclosure or suggestion of calculating a volatility metric representing an amount of code change that has occurred between software modules between one or more builds stored in a database and software module information of software being executed or between a first build and a second build registered in a database, as recited by Applicants. Specifically, Applicants have recited that determining the volatility metric includes determining at least one of: a number of functions added, a number of functions removed and a number of functions modified in software module of software being tested in comparison to a software module of the first build registered in a database. (See, for example, page 34, line 16 to page 36, line 8 of the originally-filed specification.)

The Office Action, on page 15, paragraph (C), suggests the lack of specifics as the nature of the metrics being previously-recited by Applicants provided for a broad interpretation of the term to encompass the collection of data proposed by Leblang. The Office Action cites to Leblang's use of the term delta as a metric, and Applicants note that Leblang describes storing text files as deltas (see, for example, col. 2, lines 49-51 of Leblang). However, in contrast to Leblang, as discussed above, Applicants have clarified the determination of the volatility metric recited by Applicants in the amended claims herein, specifically including determining at least one of: a number of functions added, a number of functions removed and a number of functions modified in a software module of the software module information about the software being tested, or of a second build in the database, in comparison to a software module of the one or more builds included in the database.

Accordingly, Applicants respectfully submit that Leblang does not teach or fairly suggest at least the above-noted features as claimed by Applicants. In view of the above, Applicants respectfully request that the rejection be reconsidered and withdrawn.

The rejection of Claims 11-15 and 32-36 under 35 U.S.C 103(a) as being unpatentable over Leblang is hereby traversed and reconsideration thereof is respectfully requested in view of the amendments to the claims contained herein.

The features of independent claims 1 and 22, as amended herein, are discussed in detail above with respect to Leblang. Claims 11-15 and 32-36 depend therefrom. As discussed,

Applicants submit that Leblang does not teach or fairly suggest at least the above-noted features of the claims as amended herein. Accordingly, Applicants respectfully request that this rejection be reconsidered and withdrawn.

Based on the above, Applicant respectfully requests that the Examiner reconsider and withdraw all outstanding rejections and objections. Favorable consideration and allowance are earnestly solicited. Should there be any questions after reviewing this paper, the Examiner is invited to contact the undersigned at 508-898-8603.

Respectfully submitted,
MUIRHEAD AND SATURNELLI, LLC

Date: October 30, 2007

Donald W. Muirhead
Reg. No. 33,978

Muirhead and Saturnelli, LLC
200 Friberg Parkway, Suite 1001
Westborough, MA 01581
Tel: (508) 898-8601
Fax: (508) 898-8602

**Customer No. 26339**